UNITÉ DE RECHERCHE
NRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1346

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

EXPRESSION DES RELATIONS ET MAINTIEN DE LA COHERENCE : LE CONCEPT DE LIEN.

Mireille FORNARINO
Anne-Marie PINNA

Décembre 1990

RR-1346

Expression des relations et maintien de la cohérence : le concept de lien.

Relation expression and consistency management : the link concept.

Mireille Fornarino - Anne-Marie Pinna

Résumé

Il est de plus en plus nécessaire de définir et de maintenir des relations entre objets. Mais l'expression des relations est souvent compliquée dans les langages à objets traditionnels et les mécanismes proposés pour les établir ne permettent pas d'exposer clairement les aspects déductifs qu'elles impliquent.

L'idée est d'utiliser l'approche objet pour décrire un langage de relations qui facilite l'expression des relations de dépendance et le maintien automatique de leur cohérence. Une des originalités de cette proposition est la prise en compte des modifications intervenant sur les objets maîtres pour rétablir de façon incrémentale la cohérence sur les objets esclaves.

Une hiérarchie de liens prédéfinis est proposée à l'utilisateur, qui peut ainsi définir ses propres classes de liens par spécialisation. A partir de cette hiérarchie et par extension, nous avons realisé plusieurs applications dont une modélisation des liens d'héritage des langages à objets, une interface graphique pour le langage Othelo, la gestion de la cohérence d'une bibliothèque de modèles en thermique et l'implantation d'un système d'aide pour Smeci.

Mots clés

Relations, maintien de la cohérence, représentation par objets des relations.

Abstract

The ability to define and manage relations between objects is more and more needed. But the expression of relations is often limited and tricky and the proposed mechanisms don't exhibit clearly the implicit deductive aspects hidden behind the relationships. We propose a uniform approach for relationships, where the knowledge representation for relation expression facilitates consistency maintenance and implies automatical deductions.

The idea developed in this report is the use of the object oriented programming to describe a relationnal language. So the relation expression is easier than in traditionnal object oriented language and the consistency management is automatic. In particular, we take into account the modifications on the master objects to reestablish incrementaly consistency on the slave objects.

A predefined link hierarchy is proposed to the user, who can define his own link classes by specializing existent ones. By extension of this hierarchy, we have modelised the inheritance mechanism of object oriented languages, realised a graphical interface for the Othelo language, managed the consistency of a model library in building energy, and implemented a help system for Smeci.

Keywords

Relations, consistency management, object oriented representation of relations.

Table des matières

Chapitre 1

Introduction

Généralement les relations proposées par les langages à objets [SB86] sont des relations prédéfinies et donc limitées. Le plus souvent, il s'agit de relations d'héritage *is-a* et *kind-of* [Bra83].

Or de plus en plus d'applications [Tro88,Bor86,Dug87] nécessitent l'expression d'autres sortes de relations telles que les relations simples et plus généralement les dépendances entre entités (la fenêtre F représente l'objet O, l'objet O est posé sur l'objet C, la table est composée d'un plateau et de quatre pieds, etc.) Aussi, les langages dédiés à des domaines d'applications particuliers (CAO, interfaces) introduisent-ils des relations d'agrégation[BC87], de déduction [HM87], etc.

Dans cette optique, il existe de plus en plus d'outils pointus de gestion de relations utilisant des algorithmes puissants de propagation de contraintes dans des graphes de dépendance [Col89,Gro88,Ber88].

L'objectif du travail présenté ici est la modélisation du concept général de relations de dépendance (relations orientées qui établissent des rapports de maître à esclave entre des objets donnés). Pour cela, nous avons étudié le mécanisme des relations et déterminé les différents processus qui le composent. Le modèle proposé décrit ce mécanisme au moyen du concept objet. Il permet d'exprimer simplement la sémantique relative à une relation donnée dans une seule entité et offre un mécanisme puissant de gestion de la cohérence des relations en introduisant la notion de *point d'activation*. L'originalité de cette notion est de permettre à l'utilisateur d'exprimer l'ensemble des modifications remettant en cause la cohérence d'une relation et les actions à réaliser pour rétablir cette cohérence.

Une implantation de ce modèle a été réalisée en Othelo[1] [FP90a]. Sa mise en oeuvre en terme d'objets se concrétise par des classes de lien. Celles-ci explicitent la sémantique des relations qu'elles modélisent tandis que leurs instances représentent des relations effectives entre deux objets. Les familles de liens établies à partir de ce concept constituent une hiérarchie : bibliothèque extensible de relations dans laquelle l'utilisateur puise pour définir ses propres relations.

Une telle intégration des relations étend la puissance descriptive des langages à objets tout en préservant le principe essentiel d'encapsulation des données et l'extensibilité et l'uniformité de ces langages[DG87]. Nous avons prouvé l'utilité et l'originalité de ce modèle relation-

[1] Othelo, implanté en Prolog, est un langage de schémas à comportements logiques dans la lignée de Lap [IK87].

nel dans la compréhension et la résolution de problèmes importants comme la réalisation d'interfaces graphiques cohérentes, l'expression du mécanismes d'héritage des LOO et la réalisation d'un système d'aide pour Smeci [SME88].

Ce rapport se compose de quatre parties. Tout d'abord nous situons le concept de lien vis-à-vis de mécanismes préexistants dans les langages à objets pour représenter les relations. Puis nous présentons succintement la modélisation de ce mécanisme au moyen de classes et métaclasses. De nombreux exemples de classes de liens étayent ensuite la compréhension du modèle. Un aperçu des principales applications réalisées à ce jour au moyen des liens est ensuite donné.

Chapitre 2

Représentations des relations dans un langage à objets

Dans les langages à objets, les relations d'héritage *instance* et *est-un* ou *sorte-de* sont à la base de la structuration des données; ces relations sont alors prédéfinies et leur comportement est donc figé. Une utilisation de ces langages dans certaines applications telles que la CAO et la représentation des connaissances a mis en évidence une autre relation importante qu'est le lien *partie* ou *agrégat* [FTK*84,BS83,BC87,MK87,SB86,Dug87].

Lorsqu'un utilisateur doit représenter des relations différentes de celles initialement conçues par le modèle, soit il parvient à détourner les relations d'héritage prédéfinies de leur sémantique initiale (spécialisation pour de la généralisation ou des relations de parties [Bra83, Sny86,Car84,Ame87]), soit il les modélise au moyen d'attributs, démons et/ou valeurs actives.

Dans cette partie, nous présentons brièvement les avantages et les inconvénients de l'intégration des relations par attributs, par valeurs actives et par classes.

2.1 Attributs d'objets : une façon de représenter les relations.

Dans les langages de schémas classiques, deux objets peuvent être liés en utilisant les attributs d'objets. Si la valeur d'un attribut peut être un objet quelconque, il est simple de référencer un objet dans un autre objet. L'attribut joue alors le rôle de relation et des attachements procéduraux gèrent la cohérence de la relation. Aucun concept nouveau n'est introduit et la représentation des connaissances est uniforme. Cependant, cette solution présente certains inconvénients.

- L'information concernant une relation est répartie dans toutes les classes d'objets qui peuvent être liés par cette relation. La connaissance relative à une relation est ainsi redondante et difficilement accessible.

- Pour implanter une relation, l'utilisateur doit définir l'attribut la représentant et mettre en oeuvre au moyen de démons la gestion de la cohérence. La sémantique d'une relation est alors mal localisée; elle est répartie à la fois dans l'attribut et dans les démons gérant sa cohérence. La prévention des cycles est alors d'autant plus difficile que le nombre de relations modélisées est élevé.

- L'ajout ou le retrait d'une relation entraîne des modifications de la structure des classes concernées et la redéfinition de certains démons. La spécification de relations entre instances doit être prévue par la classe des objets influant sur ces relations.

- Les démons ont diverses fonctions (vérification de types, affichage, initialisation, etc.). La gestion de la cohérence d'une relation n'est donc pas isolée d'autres fonctionnalités.

Les relations ne sont pas explicites et il est difficile d'exprimer clairement leur sémantique et leurs caractéristiques. Les aspects modulaires et évolutifs des langages à objets sont ainsi altérés.

2.2 Les valeurs actives sont des relations cachées.

Les valeurs actives qu'offrent certains générateurs de systèmes-experts comme Smeci [SME88], Kee [KEE85] et des langages de représentation des connaissances comme Loops [BS83] peuvent être considérées comme des relations cachées.

Le principe de base repose sur l'association d'objets appelés *activités* à un objet que l'on veut rendre actif et sur des envois de message à ces activités avant et après toute affectation à l'attribut de l'objet actif. Les valeurs actives offrent ainsi la possibilité d'effectuer automatiquement des actions à chaque modification de l'objet.

En conjugant l'utilisation des attributs d'objets et des démons, le mécanisme des valeurs actives permet de lier des objets, mais la relation n'est pas explicite. En effet, les valeurs actives modélisent l'activité d'un objet et non pas nécessairement des relations. Ainsi elles apparaissent comme des relations cachées dans lesquelles les objets dépendants ne sont pas clairement spécifiés.

Ce mécanisme est utilisé en particulier pour modéliser des relations de type objet-représentation graphique [DP87] mais ne permet de lier qu'un attribut à un objet d'où un déclenchement pour le maintien de la cohérence figé en général à l'affectation. Or il est souvent utile de déclencher un tel mécanisme lors de l'appel d'autres méthodes telles que l'ajout ou le retrait de nouvelles informations, la création ou la destruction d'instances, etc.

2.3 Modélisation des relations par les classes

Les recherches visant à représenter les relations par des classes se sont développées dans des domaines divers comme les bases de données et les systèmes experts. Ainsi, plusieurs études ont conduit à une modélisation objet des relations des bases de données, qui sont des relations "naturelles" [IK87], et non des relations de dépendance [BPR88,EWH85].

La démarche des générateurs de systèmes experts Art et Knowledge Craft (KC) [LTAZ87] vise à une expression explicite des relations entre objets. Ainsi l'utilisateur peut définir ses propres relations sous forme de schémas et y associer différentes informations. En particulier, il lui est possible de préciser des relations d'héritage particulières en spécifiant quelles propriétés doivent être héritées d'un objet à l'autre et de quelle manière [Cla85]. Les relations sont alors des objets comme les autres, que l'on peut spécifier et dont on peut exprimer la sémantique dans le cas des relations d'héritage. Ainsi il devient possible de réaliser des actions directement sur les relations et non pas sur les objets en relations.

Les réflexes d'OKS [Voy89] sont des objets Smalltalk qui réagissent aux événements pour lesquels ils sont programmés. Ainsi ils permettent d'exprimer des relations et d'en maintenir automatiquement la cohérence. En OKS, les réflexes sont les seuls modes de représentation

des connaissances dynamiques. Les évènements espionnés sont des modifications quelconques de l'environnement Smalltalk.

Notre démarche a donc consisté à structurer les relations sous forme d'objets dans lesquels sont regroupées toutes les informations qui les concernent et à proposer un mécanisme de maintien de la cohérence proche conceptuellement des réflexes.

2.4 Synthèse

L'ensemble des critiques faites dans ce chapitre se résume, en fait, en un seul mot "programmation". On demande aux programmeurs d'implanter les relations autres que celles prédéfinies à partir des outils du langage. Cette approche implique de détourner les concepts initiaux et complexifie ainsi à la fois l'expression et l'analyse de la connaissance relationnelle. De plus, cette programmation n'est pas facilement évolutive; il est difficile d'ajouter de nouvelles relations et elle ne permet pas une réelle factorisation de cette forme de connaissance.

L'approche qui consiste à implanter les relations sous forme d'objets organisés en une hiérarchie de classes présente par contre de nombreux avantages. Elle permet de regrouper l'ensemble des informations concernant les relations dans un même objet. Si notre approche s'inspire de cette idée, elle diffère des travaux exposés ci-dessus, notamment par la nature des relations traitées. Nous nous intéressons à l'expression de relations de dépendance qui représentent un lien de maître à esclave entre objets. Il s'agit donc de modéliser les relations de façon générale, les relations d'héritage n'en étant qu'un cas particulier.

En définissant des classes de relations, nous donnons un support sémantique à la notion de relation. Dans une classe sont regroupées toutes les informations intrinsèques à une relation et le mécanisme de maintien de cohérence est totalement régi par l'ensemble des méthodes de cette classe. Parmi elles, se trouvent les méthodes essentielles de création, de destruction et d'activation des relations individuelles.

Chapitre 3

Concept de lien

L'étude de diverses relations de dépendance nous a conduites à dégager un mécanisme général de gestion de la cohérence de ces relations.

3.1 Mécanisme des relations

Les relations étudiées ici établissent des rapports de maître à esclave entre les objets.

Dans une relation individuelle, les objets asservis sont appelés *objets dépendants* et les autres *objets influants*. Dans un souci de simplification, nous limiterons la présentation du mécanisme de maintien des dépendances à une relation entre deux objets. Dans la partie 4.4 nous verrons comment ce mécanisme peut être étendu à plusieurs objets influants et dépendants.

Lorsque deux objets sont mis en relation, l'objet dépendant doit parfois être modifié, pour que la *cohérence* de la relation individuelle soit établie.

La déclaration de la relation individuelle *carre1 a-le-même centre-que cercle1* implique de calculer la position du cercle en fonction de celle du carré.

Lorsque l'objet influant d'une relation individuelle est modifié, l'objet dépendant doit parfois être réajusté afin de rétablir la cohérence de la relation. Trois comportements sont alors possibles :

- la modification de l'objet influant ne remet pas en cause la cohérence de la relation individuelle.

  Soit la relation individuelle *carre1 a-le-même-centre-que cercle1*. Si l'objet carre1 reçoit le message "agrandir(carre1)", cercle1 n'est pas concerné par cette modification.

- la modification de l'objet influant remet totalement en cause la cohérence de la relation individuelle.

  Si l'objet carre1 reçoit le message "deplacer(carre1,10,20,45)", la relation individuelle *carre1 a-le-même-centre-que cercle1* sera valide si les coordonnées du centre de cercle1 sont recalculées.

- le changement survenu dans l'objet influant incrimine en partie la cohérence de la relation individuelle. Dans ce cas, la cohérence est maintenue de façon ponctuelle et le réajustement de l'objet dépendant est réalisé en conséquence.

  Si l'objet carre1 reçoit le message "translater-axe-x(carre1,20)", la relation individuelle *carre1 a-le-même-centre-que cercle1* sera valide si l'abcisse du centre de cercle1 est recalculée.

De l'étude des mécanismes des relations, nous avons dégagé la classe de lien que voici.

3.2 Implantation : Classe lien

Une relation donnée est représentée par une classe dont les instances sont des relations individuelles qui se comportent toutes de la même façon. La classe *lien* définit les mécanismes nécessaires à l'établissement de la cohérence des relations individuelles, tandis que la métaclasse *métalien* définit les méthodes de création et de destruction des relations individuelles. Toute classe représentant une relation est une spécialisation directe ou indirecte de *lien* et une instance de *métalien*. Nous appellerons ces classes *des liens* et les instances de ces classes *des liens individuels*.

L'implantation actuelle des liens a été réalisée en Othelo, langage à objets au dessus de Prolog. Aussi les méthodes sont-elles spécifiées en Prolog.

La métaclasse *métalien* définit les propriétés suivantes :
Attributs-d-instances : points-d-activation
Méthodes-d-instances : créer, détruire

Un lien individuel est caractérisé par sa classe, un objet influant et un objet dépendant. La classe *lien* définie les propriétés :
Attributs-d-instances: objet-influant, objet-dépendant
Méthodes-d-instances : cohérence, activer

Aussi pour un lien donné, l'utilisateur doit-il définir les propriétés suivantes :

coherence(lien-individuel) Cette méthode qui précise la sémantique de la relation est exécutée à la création d'un lien individuel. Le corps de la méthode *cohérence*, associée à un lien, définit l'action qui établit la cohérence de chaque instance.
La méthode de cohérence du lien *a-le-même-centre-que* est définie par:

```
coherence(Lien-Individuel) :-

    valeur(Lien-individuel, objet-influant,Un-carré),
    valeur(Lien-individuel, objet-dépendant,Un-cercle),
    calculer-centre(Un-carré,X,Y),
    affecter(Un-cercle,abscisse,X),
    affecter(Un-cercle,ordonnée,Y).
```

points d'activation Ce champ spécifie quand la cohérence d'une relation individuelle doit être rétablie et quelles actions il faut réaliser pour cela. Un point d'activation est constitué d'une liste de sélecteurs de méthodes[1](quand faut-il rétablir la cohérence d'une relation?) et d'une procédure de mise à jour (comment faut-il la rétablir?). Celle-ci est appelée *action* et prend pour arguments un lien individuel et un message.

Soit la classe *point-d-activation* définie par :
Attributs-d-instances: selecteurs.
Méthodes-d-instances : action, activer.

Soit le point d'activation *déplacer-p-a* :
sélecteurs : [déplacer(carré,dX,dY,Angle), translater(carré,dX,dY)]

[1]Un sélecteur de méthode est constitué du nom de la méthode, de son arité et éventuellement du type de ses arguments.

action : recalculer-centre *qui déclenche la fonction de cohérence associée au lien individuel.*

La classe du lien *a-le-même-centre-que* a alors entre autres points d'activation *déplacer-p-a.*

Les méthodes indispensables à la mise en oeuvre du mécanisme de maintien de la cohérence ne sont pas figées, et un utilisateur avancé peut les redéfinir pour en spécialiser le comportement.

Lorsqu'un lien individuel est déclenché, il y a exécution de la méthode *activer* définie dans sa classe. Cette méthode prend pour argument le message qui a causé l'activation du lien individuel. Par défaut, si le message correspond à un sélecteur reconnu par un point d'activation, l'action associée à ce dernier est alors exécutée.
Un utilisateur avancé pourra redéfinir la méthode *activer* pour, par exemple, retarder une activation ou comme nous le verrons vérifier des conditions avant l'activation (cf. §4.1).

Une instance d'un lien est créée par une méthode *créer* qui établit un lien individuel entre *objet-influant* et *objet-dépendant* passé en argument et déclenche la méthode *cohérence* définie dans le lien.

Le nombre important des relations individuelles à modéliser nous a conduites à minimiser la structure de ces objets, d'où la nécessité de la métaclasse *métalien.* En particulier dans cette mise en oeuvre, nous avons exploité au maximum les possibilités relationnelles de Prolog.

En Othelo, la représentation interne d'un lien individuel de la classe l entre les objets oi et od est le fait l(oi,od).

Chapitre 4

Des exemples de classes de liens

Des liens de dépendance plus précis ou un peu différents de ceux modélisés par la classe *lien* sont souvent nécessaires. Ainsi, en particulier lors de la réalisation des différentes applications présentées au chapitre 5, nous avons été conduites à définir plusieurs sous-classes de *lien*. Entre autres, les principales familles de liens de dépendance actuellement implantées sont :

les liens conditionnels : lors de modifications de l'objet influant, la cohérence de ces liens n'est rétablie que sous certaines conditions (cf. §4.1).

les liens sur propriétés : les dépendances concernent parfois seulement certaines propriétés des objets en relation.

les liens entre liens : l'établissement de relations entre liens permet de déduire de nouvelles relations ou de détecter des incohérences.

les liens à influants et/ou dépendants multiples : une extension du concept de lien présenté jusqu'ici consiste à introduire plusieurs objets comme influants et/ou dépendants d'une relation.

Les liens à automates : un automate est associé au lien. Selon les différentes activations du lien, l'automate change d'état, ce qui peut pour certains états aboutir à une modification des objets dépendants. Ces liens permettent en particulier de forcer le séquencement de certaines actions.

4.1 Liens conditionnels

Le comportement dynamique des liens de dépendance est défini grâce aux points d'activation. Ceux-ci permettent de préciser selon quelle modification de l'objet influant rétablir la cohérence et comment. Le principe de base du point d'activation est le suivant : si l'objet influant est modifié par l'application d'une méthode, une action est déclenchée pour rétablir la cohérence du lien : "*si telle modification alors telle action*".

Or, dans de nombreux cas, une conditionnelle apparaît indispensable. Elle permet d'exprimer non seulement quelle modification implique le rétablissement de la cohérence, mais également sous quelle condition, celui-ci doit être réalisé.

Deux sortes de conditions se distinguent. Une exprime que "sous telle condition, s'il y a telle modification alors action". Une autre dépend davantage de la nature de la modification; "s'il y a telle modification et que telle condition est respectée alors action".

Dans le premier cas, nous parlons de façon générale de *lien conditionnel*, dans le second de points d'activation conditionnée.

Lien conditionnel La méthode *activer* associée à un lien conditionel vérifie qu'une condition, exprimée sous la forme d'une méthode *condition* associée au lien, est bien vérifiée puis active le point d'activation concerné par la modification. Si la condition n'est pas vérifiée, aucun point n'est activé.

> Un lien entre un objet et une fenêtre ne doit être activé que si la fenêtre est active.

Point d'activation conditionnée Les points d'activation conditionnée permettent d'exprimer qu'une modification n'implique un rétablissement de la cohérence que sous certaines conditions.

> Soit un lien qui visualise un objet cercle à l'écran. Selon la précision du graphique, si les modifications effectuées sur le centre (déplacer) ou sur le rayon (agrandir) du cercle sont minimes, il est inutile de redessiner le cercle. Il faut donc comparer l'écart de valeur au degré de précision graphique avant de redessiner le cercle à l'écran.

La classe *point-activation-conditionnee* est définie comme une sous-classe de la classe *point-activation*. La méthode *condition* lui est ajoutée; elle prend pour arguments le lien individuel concerné et la modification qui a occasionné l'activation du point.

La méthode *activer* associée à ces points d'activation commence donc par vérifier la condition avant d'exécuter l'action.

Ces deux approches des liens conditionnels sont complémentaires. Très proches l'une de l'autre, la première factorise la condition au niveau du lien tandis que la seconde ne concerne que certains points d'activation. Les liens conditionnels sont très voisins des réflexes de OKS [Voy89], si ce n'est que l'action agit toujours sur un objet. Ils permettent donc de faire de la programmation par réflexes.

4.2 Lien de dépendance de propriétés

Prenons l'exemple de la relation *est-pere* qui unit un père à son fils, et qui détermine le nom et l'adresse du fils en fonction de celle du père. Cette relation ne concerne donc que les attributs *nom* et *adresse* d'un objet *personne*. La mise à jour d'une relation entre un père et un fils ne doit être effectuée que pour une affectation d'un des attributs *nom* ou *adresse*. Pour cela, il faut préciser au niveau de la connaissance du lien *est-pere* la liste des attributs concernés. Cette liste est alors prise en compte au moment de l'activation.

Dans la classe *lien-propriétés*, ces propriétés sont précisées par la méthode *proprietes-concernees*. L'activation d'un lien individuel prend en compte cette liste et n'est réalisée que si la modification de l'objet influant porte directement sur une de ces propriétés. Les liens de dépendance de propriétés sont donc des liens conditionnels dont la condition vérifie que la modification est réalisée pour une des propriétés concernées.

Les liens d'héritage sont des liens de dépendance de propriétés. L'héritage concerne un ensemble complet ou un sous-ensemble des propriétés d'un objet. Il est alors fréquent d'avoir à déterminer dynamiquement la liste des propriétés dépendantes : les propriétés concernées

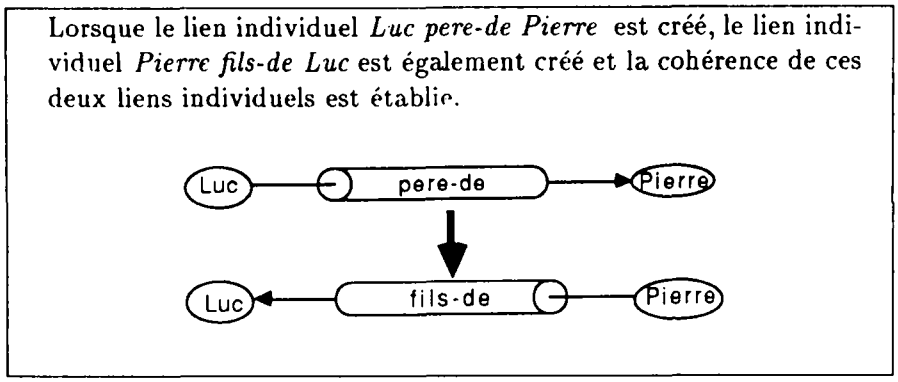


Figure 4.1: Création automatique d'un lien individuel *fils-de*.

par un lien *super* évoluent en fonction des ajouts et des retraits des propriétés de l'objet influant.

4.3 Dépendance de liens

A partir d'un réseau de liens de dépendance établi par le programmeur, il est intéressant de déduire automatiquement de nouvelles informations, telles que des liens individuels et de détecter des erreurs (incohérences).

> Si l'on sait que *Paul est-pere-de Luc*, *Luc est-fils-de Paul* est une conséquence naturelle de l'affirmation précédente. De plus, si *Paul est-pere-de Luc*, il est incohérent de dire que *Paul est-fils-de Luc*.

Ces exemples illustrent des raisonnements courants basés sur des connaissances entre relations. De cette constatation, nous avons dégagé des liens de dépendance entre liens. On pourra qualifier les relations *fils-de* et *pere-de* de relations inverses et les relations *pere-de* et *grand-pere-de* de relations incompatibles.

Les classes de liens correspondant à ces deux cas sont présentées plus dans le détail dans les paragraphes suivants.

4.3.1 Lien inverse-de

La classe *inverse-de* permet d'exprimer qu'un lien est l'inverse d'un autre.

> la relation *est-pere-de* est l'inverse de *est-fils-de*. les relations *a-meme-centre-que* et *inverse* sont leur propre inverse.

Deux caractéristiques de cette classe de lien sont à noter.

1. Lorsqu'une instance d'un lien est créée, si ce lien a un inverse, le lien inverse individuel est automatiquement créé (cf. figure 4.1).
2. Lorsqu'une instance du lien *inverse-de* est créée entre deux classes de lien, son inverse est également créé, puisque l'inverse du lien *inverse-de* c'est lui-même (cf. 4.2).

Si l'on déclare *pere-de inverse-de fils-de*, le lien individuel *fils-de inverse-de pere-de* est automatiquement créé.

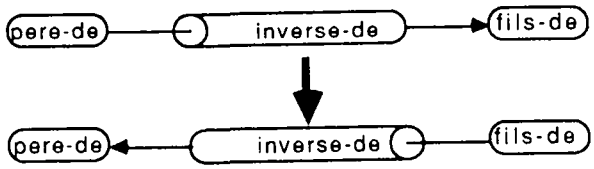


Figure 4.2: Création automatique d'un lien individuel *inverse-de*.

Il est impossible de créer une instance du lien *est-inférieur-à* s'il existe déjà une instance du lien *est-supérieur-à* reliant les mêmes objets.

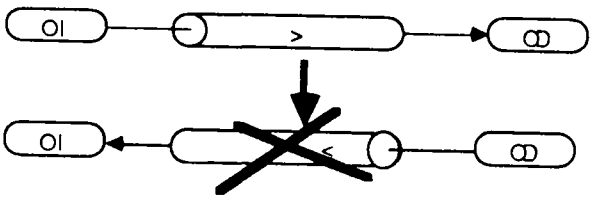


Figure 4.3: Refus de création d'un lien individuel *est-inférieur-à*.

4.3.2 Lien incompatible

La classe *incompatible* permet d'exprimer l'incompatibilité de deux liens de dépendance. Elle signale donc des incohérences lors de l'établissement de liens individuels : deux objets ne peuvent pas être simultanément liés par deux liens déclarés *incompatibles*.

Les liens *être-parallèle* et *être-perpendiculaire* sont incompatibles.
Les liens *plus-grand-que* et *plus-petit-que* sont incompatibles.

Notons que :

1. Il est impossible de créer une instance d'un lien s'il existe déjà une instance d'un lien avec lequel il est incompatible reliant les mêmes objets (cf. 4.3).

2. Le lien *incompatible* est inverse de lui-même, nous parlons alors de lien bidirectionnel. Lorsqu'une instance de la classe *incompatible* est définie, il y a création automatique de la relation inverse (cf. 4.4).

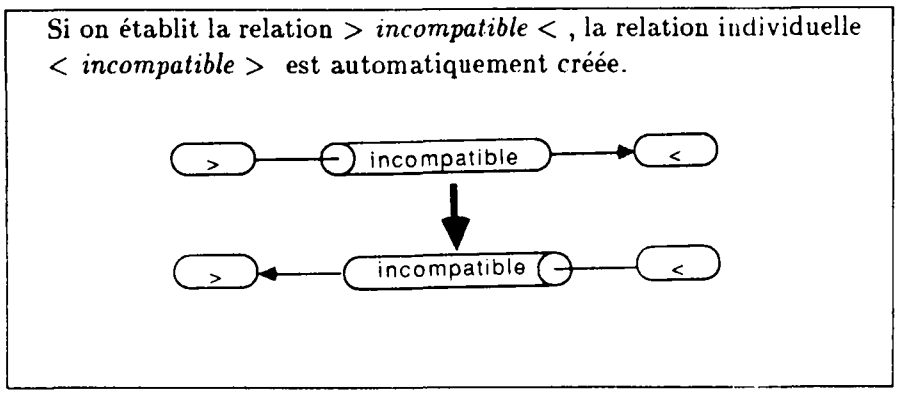


Figure 4.4: Lien incompatible - lien bidirectionnel.

4.4 Influants/Dépendants multiples

Si jusqu'ici seules des relations binaires ont été présentées, il est fréquent de constater qu'un objet dépende en même temps de plusieurs autres.

> Lorsque la contrainte x + y = z est posée, elle signifie que z dépend simultanément de x et de y. De même, la maison de Pierre est située entre celle de Jean et celle de Marie, implique bien une dépendance de position de la maison de Pierre en fonction simultanément de celle de Jean et de celle de Marie.

Les cas d'objets influants et dépendants multiples sont divers. Les relations exprimées ainsi sont des relations n-m où n représente le nombre d'objets influants et m le nombre d'objets dépendants.

Les deux principaux cas de relations à objets influants multiples sont les suivants.

1. Les objets influants participent tous à la même dépendance. Mais, pour chaque objet influant, la dépendance est vue différemment. Les objets influants ne partagent pas les mêmes points d'activation.

   > Si l'on modélise des liens d'affichage entre un objet selon un certain paragrapheur dans une fenêtre donnée, l'influence de l'objet visualisé et celle du paragrapheur, sur la fenêtre, sont différentes. Une modification de la fonction d'affichage du paragrapheur implique un réaffichage total de la fenêtre, tandis que l'ajout ou le retrait de propriétés à l'objet à visualiser aura le même effet sur la fenêtre, une modification d'une propriété de l'objet n'impliquant elle qu'un réaffichage partiel de la fenêtre.

   Dans ce cas, il y a une liste d'objets influants et une liste de points d'activation par objet influant de ce lien. La méthode *activer* cherche dans la liste des points d'activation correspondants à l'objet influant modifié, le point d'activation concerné. Si elle le trouve, elle exécute l'action associée.

2. La même dépendance unit tous les objets influants à un objet dépendant. Les points d'activation sont communs à tous les objets influants.

   > La contrainte qui unit x,y,z exprime la dépendance x+y=z. Les objets influants partagent les mêmes points d'activation; si modification de la valeur de x ou de y alors recalcul de z par x+y.

   Il y a, dans ce cas, une liste d'objets influants et une seule liste de points d'activation à examiner au rétablissement de la cohérence.

Lorsqu'une même dépendance unit un objet influant et n objets dépendants, la relation est équivalente à n relations individuelles de la même relation. Cependant, une modélisation par une seule relation clarifie la saisie de la connaissance, et peut permettre des mises à jour en parallèle ou en séquence des objets dépendants. Dans ce cas, le point d'activation implique de réaliser la même action sur tous les objets dépendants.

Un père a des fils. L'adresse de chacun dépend de la sienne.

Les relations à influants/dépendants multiples se sont révélées particulièrement utiles pour exprimer des relations multiples entre relations. En effet, il est parfois possible de déduire de nouveaux liens individuels à partir de connaissances sous-jacentes sur des compositions ou des conséquences de relations.

Les relations *(pere-de, pere-de) ou (pere-de, mere-de)* sont des *composantes-de* de la relation *grand-pere*. Les relations *(frere-de, a-le-meme-age-que)* ou *(soeur-de, a-le-meme-age-que)* sont les *causes-de* la relation *frere-jumeau-de*.

Les classes de liens correspondant à ces deux cas sont présentées dans les paragraphes suivants.

4.4.1 Lien composants-de

La classe *composants-de* permet d'exprimer qu'un lien est composé de plusieurs autres liens.

Les relations *soeur-de* et *parent-de* sont les *composants-de* la relation *tante-de*.

Certaines caractéristiques inhérentes à cette classe de lien sont :

1. Une instance du lien l (composé) est automatiquement créée entre les deux objets extrêmes dès qu'une suite de liens individuels correspondant aux liens composants (l1,..,ln) existe (cf. figure4.5).

   Un lien composé l est activé à chaque création d'une instance d'un des liens l1 ... ln. Ceci est réalisé en attachant à la classe *composants-de* le point d'activation constitué

   - de la création d'un lien individuel d'un des liens composants
   - d'une action qui crée une instance de ce lien, si toutes les conditions sont vérifiées (les instances des autres liens composants existent).

2. Un lien individuel composé est automatiquement détruit si l'un des liens individuels de la suite qui le compose est détruit (cf. figure4.6).

3. Cette classe de lien permet d'exprimer la transivité de relations telles que l'égalité (cf. figure 4.7).

Les liens *[soeur, parent]* sont les *composants-de tante*. Toute création du lien *parent* (resp. *soeur*) peut entraîner la création d'un lien *tante* si une suite de liens individuels *[soeur,parent]* est mise en évidence.

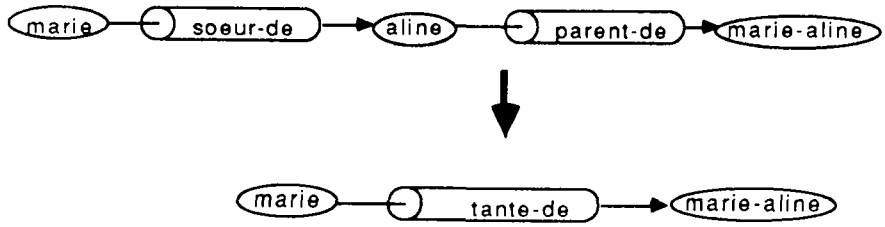


Figure 4.5: Création d'un lien *tante* individuel.

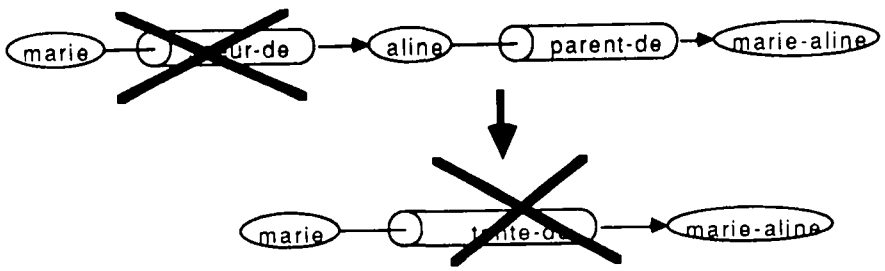


Figure 4.6: Destruction d'un lien *tante*.

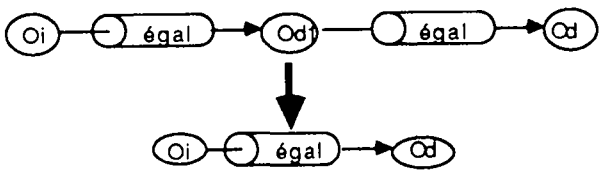


Figure 4.7: Transitivité de la relation *égal*

Un lien individuel *égal* est automatiquement créé entre deux objets s'il existe des liens individuels entre ces deux objets créés à partir des liens *inférieur-ou-égal* et *supérieur-ou-égal*, liens *causes-de égal*.

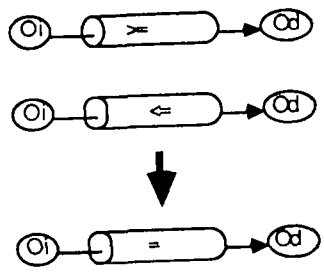


Figure 4.8: Création d'un lien individuel *égal*.

4.4.2 Lien causes-de

La classe *causes-de* permet d'exprimer l'existence d'un lien à partir de l'existence d'autres liens. On peut déduire l'existence d'un lien entre deux objets si ces deux objets sont déjà unis par d'autres liens dont il est déclaré conséquent.

La présence des liens *inferieur-ou-egal* et *superieur-ou-egal* entre deux objets donnés implique la création du lien d'*égalité* entre ces deux objets.
S'il existe à la fois un lien *a-pour-frere* et *a-le-meme-age-que* entre deux individus on peut déduire que ceux-ci sont frères jumeaux.

Cette famille de lien implique les comportements suivants :

1. Un lien individuel conséquent est automatiquement créé entre deux objets s'il existe des liens individuels des liens déclarés causes de ce lien entre ces deux objets (cf. figure4.8).
2. Il n'y a pas de notion d'ordre sur les liens l1..ln
3. La destruction d'un des liens individuels implique la destruction automatique du lien déduit (cf. figure 4.9).

Le lien *causes-de* est un lien qui exprime une dépendance entre liens

- à *objets influants multiples* : tout lien cause est un objet influant.
- *conditionnel* : il n'y a création de lien que s'il existe les liens individuels causes correspondants.

Remarques :

- Ces liens, tout comme les liens *inverse* et *incompatible* sont spécifiés de façon générale dans la classe *lien-entre-lien* dont ils sont des sous-classes. La création et la destruction de liens individuels sont des points d'activation de tels liens.

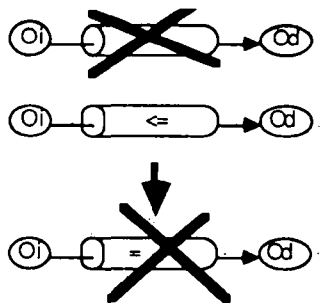


Figure 4.9: Destruction d'un lien individuel *égal*.

- Il est facile d'exprimer en Prolog des relations entre relations [Kos87], mais leur assertion automatique demande une mécanique supplémentaire. L'existence de liens n'est déductible qu'à partir d'un but à prouver.

  A l'inverse, le mécanisme des liens permet de déduire automatiquement l'existence d'autres liens et leur création immédiate.

4.5 Liens et automates

Les liens dont l'activation est conditionnée par un automate sont un cas particulier des liens conditionnels. Le principe est d'associer, à un lien de dépendance, un automate qui conditionne l'activation. Selon l'état de l'automate, il peut ou non y avoir rétablissement de cohérence. L'action à réaliser est alors non seulement conditionnée par la modification mais aussi par l'état dans lequel l'automate du lien se trouve. De sorte que l'action associée à un point d'activation agit alors sur le lien pour qu'il change d'état et éventuellement sur l'objet dépendant pour réaliser l'action de mise à jour. Les actions qui n'impliquent qu'un changement d'état du lien sont simplement des *transitions* permettant d'aboutir à un état intermédiaire. Les actions qui modifient l'objet dépendant sont des transitions qui conduisent à un état final.

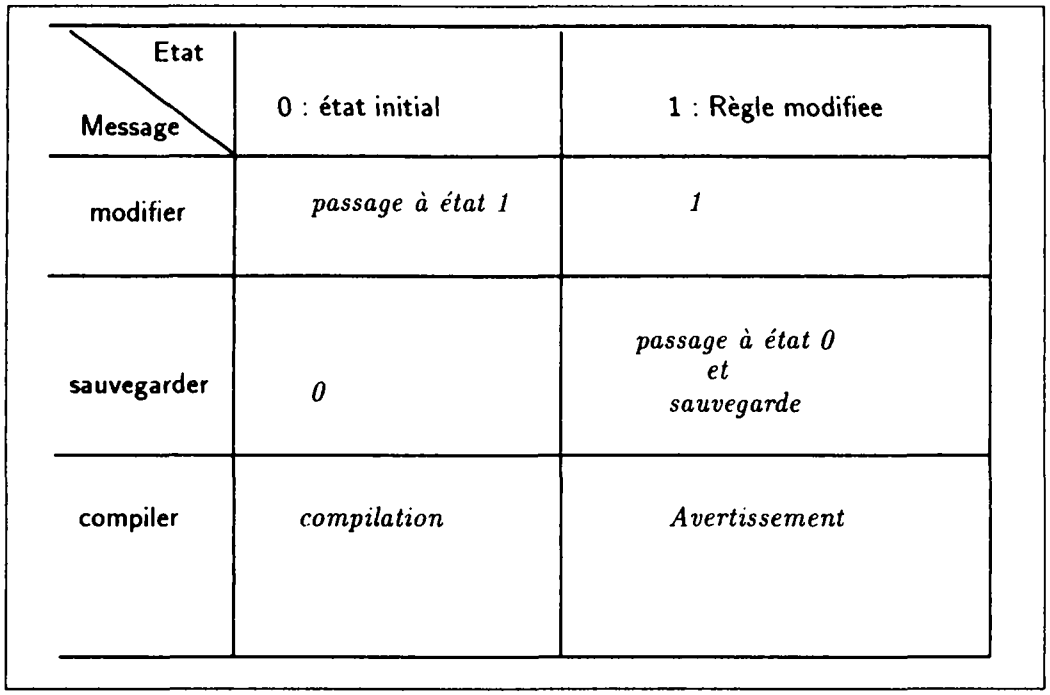
Par exemple, le lien *est-stocké-dans* est géré par un automate qui
- ne sauvegarde effectivement un objet à stocker que si celui-ci a été modifié,
- ne recompile un objet que si celui-ci a auparavant été sauvegardé.

Ceci s'exprime par l'automate de la figure 4.10.

Un tel type de lien est très utile pour exprimer la simultanéité en entrée de liens à influants multiples. En effet, il est alors possible de ne déclencher un lien que si tous les objets en entrée le demandent. Il est aussi possible de conditionner l'activation d'un lien selon l'état d'un autre lien et ainsi d'introduire une notion d'ordre sur des liens.

4.6 Conclusion

Les extensions présentées dans ce chapitre ont été facilement réalisées et enrichissent considérablement le modèle des liens. Retarder ou conditionner la mise à jour des relations résout à la fois des problèmes de cohérence et d'efficacité. De plus, ces fonctionnalités sont indispensables pour certaines applications telles que la simulation, la gestion de contraintes,



Etat / Message	0 : état initial	1 : Règle modifiee
modifier	*passage à état 1*	*1*
sauvegarder	*0*	*passage à état 0 et sauvegarde*
compiler	*compilation*	*Avertissement*

Figure 4.10: Lien à automate représentant le lien *est-stocké-dans*.

etc. Par ailleurs, nous pensons exploiter l'idée des liens à automates pour mettre à jour certains liens à partir d'un superviseur, qui déciderait quand activer les liens en attente.

Nous avons dégagé des familles de liens sous forme d'une hiérarchie de classes. Cette bibliothèque des liens est extensible et des applications particulières l'ont d'ailleurs étendue. Celles-ci sont développées en détail dans le chapitre suivant.

Chapitre 5

Des applications

Les liens se sont dès à présent révélés très utiles pour réaliser des applications importantes. Nous présentons ici, les plus avancées.

Une première application (cf. §5.1) décrit l'héritage des LOO en terme de propagation d'information et de maintien de la cohérence. Une telle modélisation de ce mécanisme autorise une multiplicité de liens d'héritage (spécialisation, instanciation, partie-de, composé-de, etc.) dont la sémantique est clairement établie.

Une deuxième application (cf. §5.2) traite de la réalisation d'une interface graphique pour langages à objets et du maintien de la cohérence entre objet et représentations.

Dans le cadre d'un contrat avec le CSTB, nous avons travaillé à la réalisation d'une modèlothèque en thermique, pour laquelle les liens se sont révélés très précieux dans la modélisation des relations entre hypothèses et au maintien de la consistance du système (cf. §5.3.2).

Une quatrième application, en cours de réalisation, est la spécification et l'implantation d'un tutoriel et d'un système de maintien de la cohérence pour le générateur de système expert Smeci. Les liens nous ont permis de modéliser dans ce cadre les relations entre concepts et entre entités utilisateurs.

5.1 Liens d'héritage

Dans les langages à objets, le lien *sorte-de* est souvent utilisé comme un mécanisme de structuration des connaissances; sa sémantique n'est alors pas clairement établie. Doit-il unir un ensemble à un sous-ensemble, signifier une spécialisation stricte ou simplement être considéré comme un outil de structuration des informations? A-t-on le droit d'exprimer des exceptions? Comment faire remonter des informations par généralisation?

Certains langages [BS83,Dug87] introduisent le lien de partie. Or selon les applications, la propagation d'information attendue de ce lien est différente. Elle est réalisée soit des parties vers l'objet composite soit de l'objet composite vers ses parties. De plus, il est nécessaire d'exprimer ce type de relation soit entre objets généraux (classes) soit entre objets particuliers (instances).

De façon générale, nous parlons de lien d'héritage lorsqu'il y a propagation d'informations sur les propriétés d'un objet vers un autre. Ainsi il pourra s'agir de lien de spécialisation, lien de partie, d'instanciation, de généralisation, de conséquence, etc.

Or une grande variété de liens d'héritage est rarement fournie par un langage. On dissocie le plus souvent deux liens : héritage et instanciation. Mais l'unique lien d'héritage est

fréquemment utilisé pour simuler plusieurs types de propagation.

Notre ambition a donc été de représenter ce mécanisme habituellement figé simplement comme une dépendance entre deux objets. De la sorte, non seulement le mécanisme n'est plus figé, mais des règles strictes quant à l'utilisation de tel ou tel lien ont pu être établies.

Pour établir un véritable lien de spécialisation, il faut n'autoriser la création d'une instance d'un tel lien que si l'objet dépendant vérifie toutes les restrictions imposées par l'objet influant (type des attributs, typage des paramètres de méthodes, ... [RM90]). Il est cependant assez difficile, d'établir l'ensemble de ces conditions de façon générale[Ame87].

Le principe que nous avons adopté est alors le suivant: l'objet *influant* d'un lien d'héritage émet via le lien des informations qui sont réceptionnées par l'objet *dépendant.* Par exemple, dans le cas d'un lien de spécialisation, l'objet influant correspond à la super-classe, l'objet dépendant à la sous-classe. L'expression de liens d'héritage permet alors un maintien automatique de la consistance entre les objets ainsi liés. Une modification d'une propriété d'un objet peut déclencher la mise à jour de l'ensemble des liens d'héritage qui ont cet objet comme objet influant et qui sont concernés par cette propriété.

Les liens d'héritage sont simplement des liens sélectifs qui concernent un ensemble de propriétés. Les propriétés concernées peuvent être précisées par une énumération ou par une méthode. Les liens d'héritage sont donc un sous-ensemble des liens *lien-propriétés.* La sémantique d'un lien d'héritage donné est ainsi complétée par la donnée de la méthode *héritage* qui prend pour arguments l'objet émetteur, l'objet récepteur et la liste des propriétés concernées. Ainsi la méthode de cohérence associée à un lien d'héritage fait appel à la méthode d'héritage pour effectuer la propagation d'informations et la méthode *propriétés-concernées* est utilisée comme un filtre. Un utilisateur peut ainsi définir, par exemple, des liens de partie en précisant quelles sont les propriétés concernées. De même il pourra déclarer des liens génériques au niveau des classes d'objets. Dans ce cas, la création d'une instance impliquera la création d'un lien individuel correspondant au lien générique.

> Soit la classe *mur-de-ma-maison*, soit le lien générique *partie-de-maison.* Si nous déclarons la classe *mur-de-ma-maison* liée par le lien générique *partie-de-maison* à l'objet *ma-maison*, alors toute instance de cette classe sera liée à l'objet *ma-maison* par le lien *partie-de.*

Lorsqu'il s'agit d'héritage dynamique, comme c'est le cas dans la plupart des langages à objets pour les méthodes, il y a également parcours des liens d'héritage. En particulier, l'envoi de message qui procède par synthèse [DH89] pour rechercher des informations dans les super-classes doit alors prendre en compte l'existence des liens d'héritage. Ainsi dans un langage tel qu'Objvlisp [Coi87] la méthode *lookup* doit être écrite pour parcourir ces liens.

Les liens d'héritage tel que nous les concevons, ne peuvent pas violer le principe d'encapsulation des données prônés par les langages à objets. En effet, le lien sert uniquement de propagateur : il s'adresse aux objets mis en relation par envoi de message pour connaitre les informations à propager. En conséquence, il reste tout à fait possible de dissimuler des propriétés en fonction des liens qui tentent de réaliser l'héritage. Ainsi, les propriétés restent tout de même responsables de la propagation des informations et peuvent propager des informations différentes selon les liens [FP90a].

Enumérons quelques avantages d'une modélisation des liens d'héritage par des liens de dépendance.

Avantages des liens de dépendance pour l'héritage

- L'expression de l'héritage sélectif.

  L'aspect sélectif de l'héritage se situe au niveau du lien où il est possible de stipuler les propriétés mises en relations (*lien-propriétés*). Un paramétrage au niveau des objets reste néanmoins possible, puisque la propagation procède par envoi de messages aux objets influants et dépendants.

- La sémantique des liens d'héritage est explicite.

  Il est possible de préciser des contraintes sur les objets mis en relation.

  > L'expression d'un lien de spécialisation implique de vérifier, à la création d'un lien individuel, certaines restrictions sur l'objet récepteur comme le type des valeurs des attributs et le typage des méthodes[Car84,Ame87,RM90].

- La mise à jour automatique des relations.

  Il suffit d'exprimer au niveau du lien quels sont les points d'activation de la dépendance pour la mettre dynamiquement à jour.

- L'expression de liens inverses est facilitée.

  > Il est possible d'exprimer que la généralisation est l'inverse de la spécialisation

Les liens d'héritage sont plus une structure d'accueil qu'une définition d'un mécanisme. C'est à leur niveau que se situe le contrôle et la propagation des information, mais le programmeur peut moduler ce comportement en attribuant des comportements particuliers aux différentes sortes de propriétés. Il peut également redéfinir les mécanismes d'héritage en créant de nouveaux liens.

5.2 Une interface graphique cohérente

Cette application a mis en évidence les relations de dépendance sous-jacentes à un environnement graphique cohérent dans lequel les objets sont visualisés de différentes façons et dans plusieurs fenêtres. Cette étude est illustrée par la réalisation d'une interface graphique pour le langage Othelo. Les objets en présence sont des objets Othelo et des objets fenêtres. L'outil graphique utilisé est le générateur d'interface Aida [Aid89], écrit en Le-Lisp. [LeL89].

Cet environnement offre les fonctionnalités suivantes :

- Un objet peut être affiché dans sa totalité, ou partiellement en montrant uniquement la liste des attributs et des méthodes. Des paragrapheurs différents guident l'affichage.
- Une propriété particulière d'un objet peut être visualisée. Par l'intermédiaire de cette représentation des modifications de cette propriété sont possibles.
- Le graphe d'héritage d'un objet peut être visualisé, selon différents éditeurs d'arbres (prise en compte, par exemple, du niveau de granularité de l'affichage).
- Un objet Othelo est compilé en clauses Prolog sous une forme que l'on peut examiner. Une telle fonctionnalité est réservée aux programmeurs avancés et peut être utilisée pour le déverminage de grosses applications. Aussi l'application graphique prend-elle en charge la visualisation des objets compilés.

- La liste des objets, avec lesquels un objet est en relation, est une information intéressante pour vérifier la validité de la connaissance saisie. L'ensemble des relations mettant en jeu un objet est donc visualisé dans un type particulier de fenêtre.

L'interface développée a un rôle de visualisation et de saisie. Les fonctionnalités offertes par cette interface suivent des règles de cohérence suivante :

1. Toute visualisation à l'écran, partielle ou totale, d'un objet doit toujours être cohérente avec l'objet sous-jacent. Si certaines propriétés d'un objet sont modifiées, toutes les visualisations qui dépendent de ces propriétés, doivent être mises à jour.
2. Il est parfois possible de modifier interactivement une représentation d'un objet. Ces modifications doivent alors être sauvegardées dans le système et, dans ce cas, être systématiquement répercutées sur l'objet correspondant dans la base de connaissance.
3. Toute affectation d'un attribut visualisé à l'écran entraîne la mise à jour de toutes les autres représentations où est mentionné cet attribut.
4. Le fait de lier un objet à un nouvel objet doit être pris en compte par une éventuelle visualisation de la liste des liens concernant cet objet.
5. Certaines modifications d'objets paragrapheur ou compilateur entraînent la mise à jour des fenêtres de visualisation ou des objets compilés concernés par cette modification. En effet, si l'affichage de certains objets est réalisé selon ce paragrapheur, il faut tenir compte de ces modifications de façon à remettre en question, si besoin, les représentations qui en dépendent. Cette fonctionnalité concerne plus particulièrement les implanteurs d'objets paragrapheurs ou compilateurs pour la mise au point de leurs outils.

5.2.1 Principales classes de liens utilisées dans cette application

Classes représentation

La classe *représentation* est une fille de la classe *lien*. Elle regroupe l'ensemble des liens qui unissent un objet à une de ses formes de représentation (graphe, édition, représentation compilée, etc.).

La méthode de cohérence d'un lien qui unit un objet à une fenêtre est généralement une méthode d'affichage, définie par un troisième objet (paragrapheur ou éditeur de graphe). Les points d'activation d'un tel lien sont des méthodes de modification de l'objet représenté, telles que l'ajout ou le retrait d'un attribut, et les méthodes de modification de la fonction d'affichage.

En effet, en cas de modification de la méthode d'affichage d'un paragrapheur ou d'un éditeur, les représentations qu'il a permis de réaliser doivent également être mises à jour.
De même, la compilation d'un objet en clauses Prolog passe par un compilateur. En cas de modification du compilateur, la représentation compilée doit être modifiée.

La signification d'un lien de représentation est : "je désire représenter tel objet par tel objet selon tel paragrapheur ou compilateur".

Les liens de représentation sont des liens 2–1, liant un objet à représenter et un moyen de représentation (paragrapheur, éditeur d'arbre ou compilateur) à une représentation.

Plusieurs sous-classes de *representation* sont définies : les classes *representation-affichage*, *représentation-compilée*, *graphe-de* sont des sous-classes de la classe *représentation*. Elles découpent cette classe en trois sous-ensembles ayant chacun une fonctionnalité différente. Une

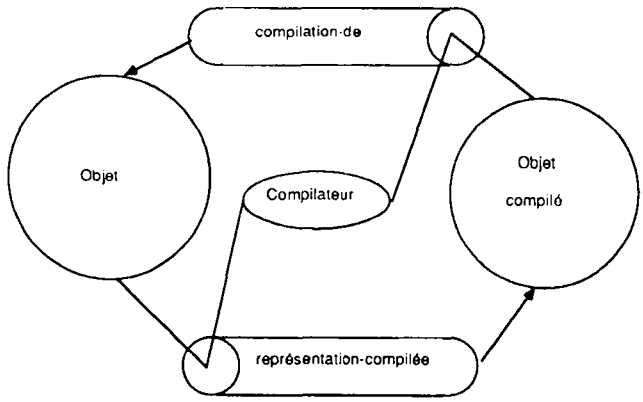


Figure 5.1: Représentation du lien *compilation-de*.

telle distinction permet, par la précision du type des objets mis en relation et par l'affectation des points d'activation, de structurer et d'identifier des liens particuliers.

Classe compilation-de

Cette classe représente l'ensemble des liens, sous-ensemble des liens de dépendance, qui unissent un objet compilé à l'objet correspondant non compilé. Cette classe représente les liens inverses de ceux de la classe *représentation-compilee*. Cependant, actuellement, cette classe est implantée indépendamment d'un compilateur : toute modification d'un objet compilé implique le rechargement automatique de la totalité des clauses représentant cette objet.

Il serait intéressant, lorsqu'une méthode de décompilation est associée à un compilateur, de créer pour chaque lien individuel instance de la classe *représentation-compilée*, le lien individuel inverse instance de la classe *compilation-de*. A ce nouveau lien individuel serait associé comme moyen de représentation le compilateur associé à son inverse.

La hiérarchie des liens du gestionnaire de fenêtres est donc représentée par la figure 5.2.

L'application présentée dans cette partie est un embryon d'interface qui doit évoluer vers la saisie d'objets et vers des liaisons entre fenêtres. Nous espérons ainsi obtenir des éditeurs d'objets avec gestion automatique de la cohérence par liens de dépendances. De même, il est parfois utile d'asservir des fenêtres entre elles pour pouvoir sélectionner, dans une fenêtre, une partie d'un objet et que celle-ci soit automatiquement sélectionnée dans toutes les fenêtres asservies représentant cet objet. A partir de cette application, nous envisageons un avenir prometteur pour les liens de dépendance dans la réalisation d'interfaces cohérentes.

5.3 Gestion de cohérence entre hypothèses

Dans le cadre d'un contrat avec le CSTB[1], nous avons été conduites à modéliser des connaissances thermiques au moyen des liens. Nous avons plus exactement travaillé à la constitution d'une sorte de librairie de modèles.

En effet, la modélisation de problèmes physiques complexes est généralement réalisée par décomposition du problème et/ou des objets à modéliser ou simuler. Cette décomposition du

[1]Centre Scientifique et Technique du Bâtiment

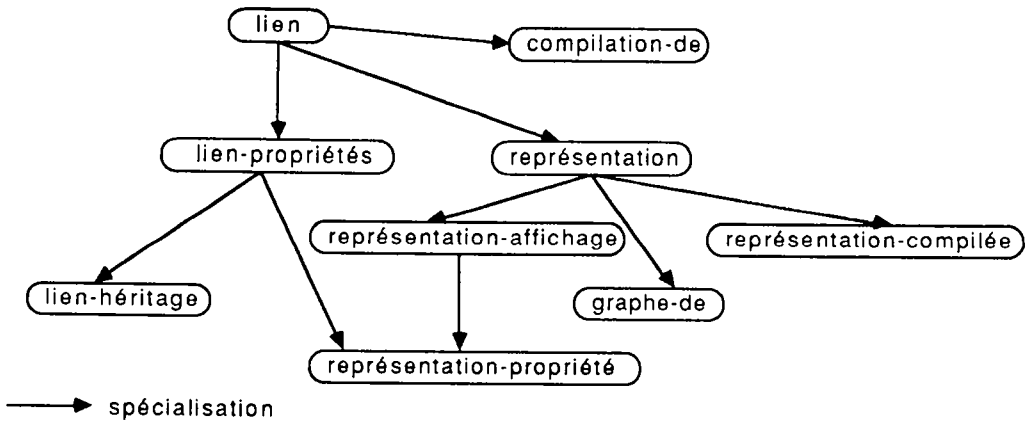


Figure 5.2: Hiérarchie des liens d'interface.

projet en éléments principaux est une part importante de la tâche de modélisation. La librairie de modèle doit justement permettre d'alléger ce travail. Elle est accessible par consultation et un expert peut la compléter. Cette librairie permet également d'associer un modèle à chaque sous-partie d'un problème. Ainsi il est possible de proposer différentes sortes de modèles élémentaires avec des hypothèses spécifiques et d'aider l'utilisateur dans ses choix en fonction des données du problème, de la confiance que l'on a dans les modèles et des hypothèses spécifiques au problème.

Ce travail peut être vu comme une première étape à la modélisation par objets des principaux concepts gravitant autour d'un modèle tels que les hypothèses, paramètres, variables, objets (un bâtiment ou un mur), phénomènes traités. Un autre point important est l'aspect intelligent de la modèlothèque quant à la sélection et l'assemblage cohérent des modèles pour répondre à un problème donné.

Dans le cadre de ce rapport, nous détaillerons plus particulièrement les relations entre hypothèses mises en jeu par cette application. En effet, la nécessité d'identifier, de sélectionner et d'assembler des modèles appropriés à la résolution d'un problème nous a amenées à gérer des ensembles d'hypothèses.
Sous ce terme d'hypothèses "supposition que l'on fait d'une chose possible ou non, et dont on tire une conséquence (petit Larousse)" nous regroupons un ensemble d'informations qui ont permis d'aboutir à la mise en place d'un modèle. Il peut s'agir aussi bien de contraintes de domaines, d'approximations, de suppositions qualitatives ou quantitatives que d'un savoir de l'expert dans un cadre donné, lorsque celui peut être mis en doute sous certaines conditions.

Or toutes les hypothèses ne représentent pas des concepts indépendants et diverses formes de relations les lient. De même les hypothèses peuvent être associées différemment aux problèmes et aux modèles.

Ainsi lorqu'un utilisateur définit un nouveau modèle, il est souvent possible de déduire à partir de données élementaires telles que le phénomène traité, des hypothèses sur ce modèle. Or l'expert peut ajouter à ce même modèle de nouvelles hypothèses et en exclure d'autres. Les liens nous ont permis de gérer la cohérence de ces différentes actions.

De même lorsqu'un utilisateur expose un problème, il le fait sous certaines hypothèses. La détermination des modèles pouvant résoudre ce problème doit alors vérifier la cohérence entre les hypothèses du problème et les hypothèses des différents modèles.

5.3.1 Association d'hypothèses à un modèle

Toutes les hypothèses associées à un modèle doivent être cohérentes entre elles, dans la mesure des liens qui les lient au modèle. Par exemple, si vous associez l'hypothèse *milieu homogène* à un modèle, vous ne pourrez pas lui associer l'hypothèse contradictoire *forte convection* mais vous pourrez l'exclure. Les liens entre hypothèses 5.3.2 et ceux qui unissent un modèle à ses hypothèses permettent de maintenir la consistance des hypothèses associées à un modèle.

Dans la réalisation actuelle, il est possible d'associer des hypothèses à un modèle par les relations : **a_pour_hypothèse, a_pour_approximation, a_pour_hypothese_indispensable, exclut_hypothese.**

Mais d'autres hypothèses non explicitement données par l'expert peuvent être déduites par le système. Il devient alors possible de distinguer différentes sortes d'hypothèses :

Les hypothèses déclarées qui sont associées directement au modèle par l'expert lors de sa définition.

Les hypothèses déduites sont ajoutées au modèle par le système. Ces hypothèses sont soit:
- héritées du modèle d'un modèle de référence [FP90b],
- déduites du phénomène sur lequel porte le modèle,

  Un calcul détaillé de la température d'une paroi implique de prendre en compte les deux surfaces de la paroi.

- déduites de la classe d'objets traitée par le modèle.

  Une paroi opaque peut être considérée comme un milieu homogène qui présente une grande absorptivité aux grandes longueurs d'ondes.

Les hypothèses explicatives ne sont jamais directement vérifiées. Il s'agit d'hypothèses qui "plus faibles" que d'autres hypothèses associées au modèle. Elles apparaissent lorsque l'expert recouvre des hypothèses existantes en associant au modèle de nouvelles hypothèses liées par implication aux anciennes hypothèses précédemment associées au modèle (par déduction, par exemple). Ces dernières deviennent alors explicatives.

Négliger les échanges peut conduire à considérer la puissance échangée comme nulle. Cette dernière hypothèse *puissance échangée nulle* peut donc devenir explicative au niveau d'un modèle impliquant l'hypothèse *Négliger les échanges.*

Il nous semble important de garder la trace de telles hypothèses au niveau d'un modèle pour faciliter sa compréhension et pour pouvoir les rétablir lors d'un retrait des hypothèses les recouvrant. Ces hypothèses ne sont pas héritées par les modèles dérivés.

L'"héritage" des hypothèses, la détermination des hypothèses explicatives est alors gérée directement par des liens.

5.3.2 Liens entre hypothèses

Une étape préliminaire à l'ajout d'une hypothèse à un modèle et à la sélection d'un modèle pour résoudre un problème est une vérification de la cohérence entre hypothèses.

Deux hypothèses sont cohérentes l'une vis-à-vis de l'autre si elles ne sont pas contradictoires. Le chemin de contradiction n'est pas toujours direct.

X implique Y contradictoire avec Z. X et Z sont donc contradictoires.

Actuellement la cohérence entre deux hypothèses données est traitée par une validation des liens qui les unissent. Nous avons dégagé les relations suivantes :

implique_contrainte : cette relation permet de préciser l'ensemble des contraintes induites par une hypothèse.
L'hypothèse Milieu transparent implique que le flux net radiatif est nul.

implique_approximation : certaines hypothèses pour être valides induisent la vérification d'approximations.
Négliger les convections permet d'approximer qu'il n'y a pas de transfert convectif avec une erreur de tant.

implique_hypothese : il s'agit en quelque sorte d'une relation d'ordre entre hypothèses. Telle hypothese n'est valide que si toutes les hypothèses qu'elle induit le sont également.
Considérer un milieu comme homogène implique de négliger la convection.

Cette relation entre hypothèses est primordiale car elle permet de diminuer les hypothèses exprimées en autorisant les regroupements.

Par exemple, l'hypothèse *milieu homogène* implique l'hypothèse *négliger la convection dans ce milieu*, qui implique l'approximation *transfert convectif nul*. L'hypothèse *milieu homogène* est donc plus générale que l'approximation *transfert convectif nul*.

est_contradictoire_avec : certaines hypothèses sont par essence contradictoires entre elles.

Ainsi nous avons actuellement les hypothèses contradictoires suivantes :
(ecoulement_irrotationnel - ecoulement_rotationnel),
(regime_permanent - regime_dynamique),etc.

est_egale_a : toute hypothèse est égale à elle même. Ce lien est utilisé lors de la validation entre les hypothèses d'un modèle et d'un problème.

est_équivalente_a : Ce lien permet de donner une représentation différente d'une même hypothèse. La définition d'un tel lien entre deux hypothèses est sujette à certaines vérifications de cohérence surtout vis-à-vis des hypothèses impliquées par l'une et l'autre. Actuellement, ces vérifications consistent à déceler d'éventuelles contradictions.
x implique y contradictoire avec z. u implique z.
Il n'est pas possible de déclarer x et u comme équivalente.

5.4 Modèle conceptuel pour Smeci

Aujourd'hui la réalisation d'un système d'ingéniérie doit débuter par une spécification de ses fonctionnalités, ce qui peut agréablement se faire par l'écriture de classes d'objets dans lesquelles le comportement de l'outil est défini. Cependant, à cause d'un manque de moyens, l'ensemble des interconnexions entre entités, bien que spécifiable au niveau des classes elles-mêmes, ne peut pas être mis en oeuvre aussi simplement et des systèmes annexes de gestion de la cohérence apparaissent, par exemple pour les éditeurs ou pour le raisonnement.

Or, les liens permettent justement d'exprimer de telles relations entre les classes représentant les concepts manipulés par l'outil décrit. De la spécification de ces liens, il est alors possible de déduire les dépendances qui devront exister ultérieurement au niveau des instances de ces concepts et ainsi de maintenir la cohérence des applications utilisant ces instances.

De la spécification d'une relation entre classes pourra découler l'existence de plusieurs dizaines, voire centaines, de relations au niveau des instances. Le gain tant pour l'implantation que pour son maintien est alors évident puisqu'il y a factorisation de l'information relative aux relations et déduction automatique de leurs existences.

Sur la base de ces idées, nous avons modélisé le modèle conceptuel de Smeci, dont nous avons déduit un système de maintien de la consistence au niveau des objets utilisateur. De plus nous avons exploité ce graphe entre concepts pour réaliser un tutoriel pour Smeci[SME88].

Ainsi, à un débutant nous proposons un graphe de relations, qui décrit le modèle conceptuel du système Smeci, qu'il peut alors interroger pour connaître les conséquences d'une action, ou savoir comment réaliser une opération.

5.4.1 Niveau conceptuel

Un premier niveau décrit les relations entre les concepts du générateur de systèmes experts Smeci, tels que *catégorie, règle de production ou tâche.* Il permet ainsi de modéliser les activités qu'un utilisateur de Smeci peut entreprendre de façon simultanée. En effet les concepts mis en jeu dans le système Smeci ne sont pas tous indépendants. Les actions de sauvegarde, compilation, etc doivent suivre un ordre strict.

> Une règle fait toujours partie d'une base de règle, qui elle est en général associée par son nom à une tâche.
> Une règle peut entraîner la création d'états.
> Un état est nécessairement généré par une règle.

Les liens nous permettent de représenter les connexions entre concepts (cf. figure 5.3 [2]).

Lorsque l'implanteur du système précise l'existence de ces liens "génériques", non seulement il met à jour la nature des dépendances entre classes mais il précise également les liens qui devront exister au niveau des instances de ces classes ou qu'il serait souhaitable d'avoir. Ainsi ces liens entre classes sont non seulement un moyen de spécifier la sémantique du futur système, mais aussi un modèle pour l'implantation de l'outil. Les liens déduits au niveau des instances constituent alors l'implantation du système.

Les utilisateurs d'un système d'ingéniérie doivent faire face à différentes difficultés selon qu'il s'agit de nouveaux utilisateurs, non familiers avec le système qu'ils doivent alors comprendre, ou d'utilisateurs expérimentés qui ont alors besoin d'une aide pour le maintien de la cohérence de leurs applications.

Les liens nous ont permis d'affronter ces deux challenges.

[2] Notons que la plupart des liens représentés sur ce graphe ont un inverse, non visible ici.

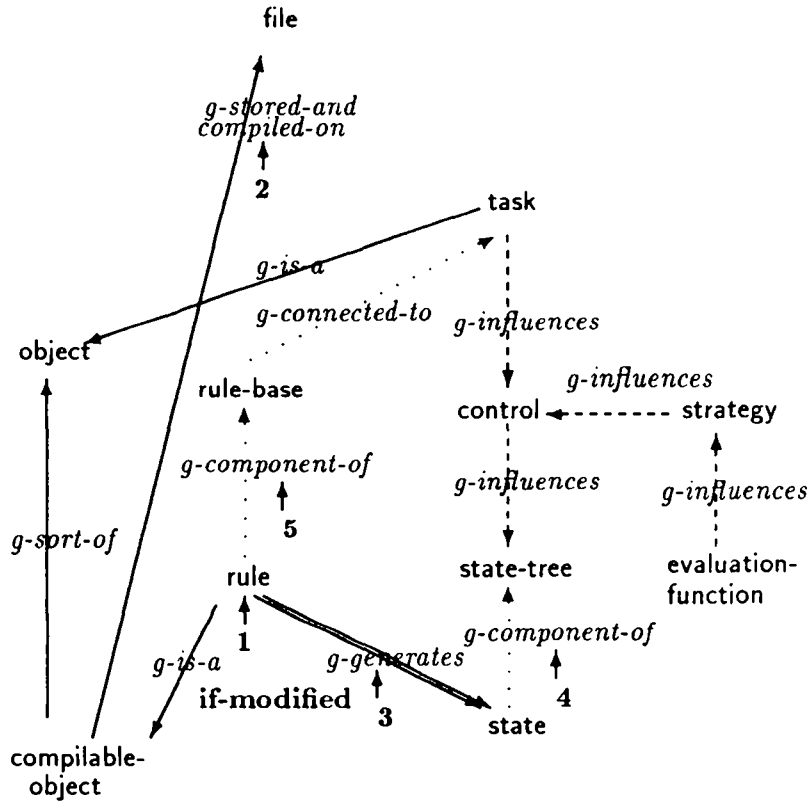


Figure 5.3: Concepts et relations au niveau conceptuel

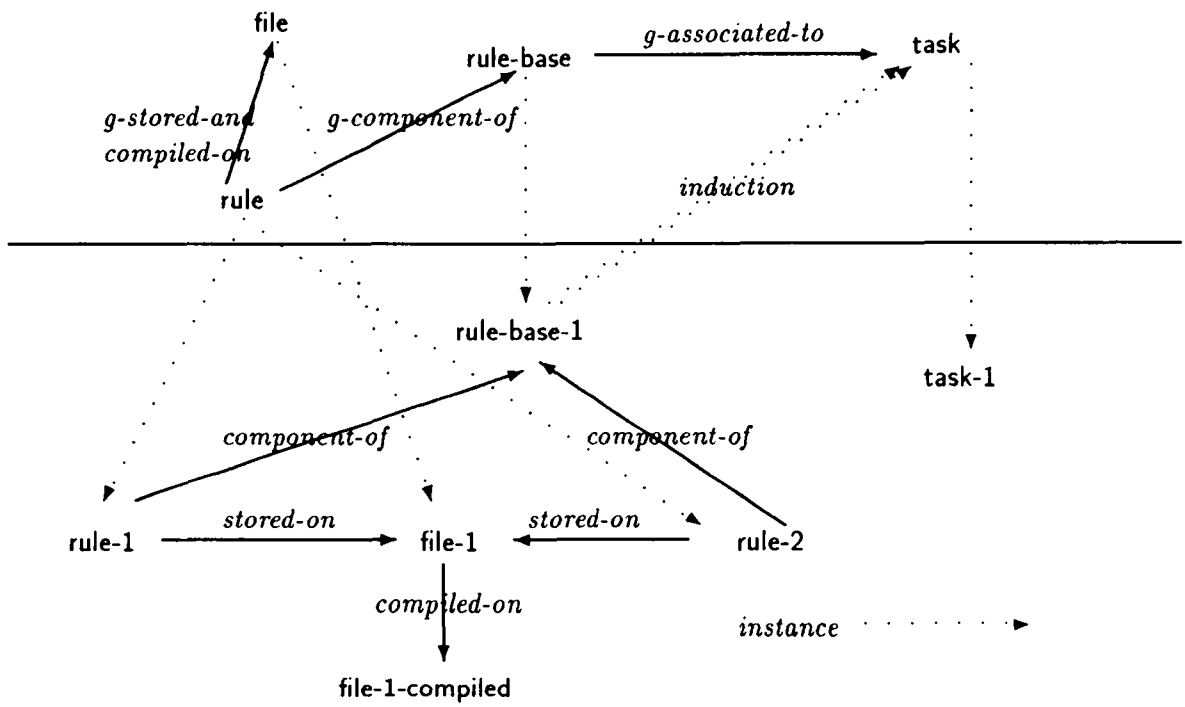


Figure 5.4: Connection entre les deux niveaux

5.4.2 Niveau utilisateur : le système de maintien de la cohérence

Le second niveau contient les instances des concepts précédents créées par l'utilisateur. Par exemple, dans une session Smeci, si un utilisateur crée deux règles, celles-ci correspondront à deux noeuds du niveau utilisateur. Les méthodes définies sur ces noeuds sont entre autres, *modifier* et *creer* qui correspondent à des actions qui seront réellement réalisées. Pour maintenir la cohérence de ces règles vis-à-vis du système, des liens déduits du niveau conceptuel doivent alors être créés. Ces liens ont la charge par exemple de forcer la sauvegarde d'une règle avant sa compilation. Ce cas est traité à l'aide de liens à automates. Ainsi, l'état des automates associés aux liens *stored-on* et *compiled-on* permet d'avertir

l'utilisateur qu'il veut compiler une règle non encore sauvegardée (cf. 4.5).

5.4.3 Connection entre les deux niveaux

Les deux niveaux conceptuels et utilisateurs sont dépendants; le niveau conceptuel est utilisé pour "générer" le niveau utilisateur et pour valider sa cohérence. Du graphe du premier niveau, nous pouvons facilement déduire la nécessité des liens entre instances. Notre objectif n'est pas de forcer la création immédiate de ces liens, mais plutôt de donner un moyen efficace de vérifier la cohérence du système utilisateur en fin de session ou sur demande. Par exemple du lien *g-associated-to* entre les classes *rule-base* et *task*, nous pouvons déduire qu'il devra exister un lien entre la base de règles *rule-base-1* et sa tâche, même si celle-ci n'existe pas encore.

Chaque fois qu'une nouvelle instance d'un concept est créée, si celui-ci est connecté à

d'autres concepts par des liens génériques, cette instance devra ultérieurement être connectée à des instances de ces derniers par des liens analogues. Cette information est stockée au moyen de liens spécifiques (dits d'induction), jusqu'à ce que l'instance soit effectivement connectée par les liens attendus. Les liens d'induction correspondant sont alors détruits. Donc, si, en fin de session, des liens d'induction existent encore, il y a incohérence dans le système, et l'utilisateur est prévenu. Ces liens permettent de plus de préciser quelles sortes de liens doivent être créés et quelles sont les instances inconsistantes.

5.4.4 Une application du niveau conceptuel : le tutoriel

Un novice face à Smeci se pose des questions comme "que se passe-t-il si je crée une instance de tel concept?" ou "comment réaliser telle action?", ce que nous avons traduit par des méthodes *si-cree*, *si-modifié*, *comment-changer*, *comment-creer*. Ces méthodes correspondent à des points d'activation des différents liens connectant les concepts entre eux. Ainsi, supposons qu'un utilisateur veuille connaître les conséquences d'une modification de règle. Il lui suffit alors d'adresser le message *si-modifie* au noeud règle (cf. figure 5.3). Puisque *g-is-a* et *g-sort-of* n'ont pas d'autres actions correspondant à ce message que de propager le message à leurs récepteurs, ce message (1) suit dans le système la séquence suivante : d'abord l'activation du lien *g-stored-and-compile-on* (2) déclare : *"la modification d'une règle implique de sauvegarder la règle sur un fichier puis de la compiler"*, ensuite par la relation *g-generates* (3) on obtient : *"une modification d'un champ différent du nom d'une règle peut induire une modification ou création d'état."* et finalement les deux relations *g-component-of* (4,5) ajoutent : *"une modification d'un état implique une modification de l'arbre d'états"* et *" une modification du nom d'une règle implique une modification dans sa base de règles"*.

Notons qu'il est nécessaire de prendre en compte la nature de la modification; par exemple des questions sur les conséquences d'une modification du nom d'une règle ou de ses prémisses ne conduiront pas à la même réponse. Les liens utilisés dans cette application sont représentés dans la figure 5.5.

5.5 Bilan des applications

Les applications présentées dans ce chapitre, nous ont permis de valider notre vision des relations de dépendance.

- La modélisation de l'héritage en terme de liens apporte à ce mécanisme plus d'extensibilité et d'abstraction. L'héritage n'est plus un mécanisme câblé global au langage.

- L'interface graphique proposée, bien que réduite, met en exergue la puissance du concept de lien. En définissant la sémantique des relations au niveau même des liens, le développement d'une interface complexe est grandement facilité. Le programmeur peut séparer l'expression des problèmes dans leur généralité des interdépendances particulières.

- La réalisation de la modèlothèque et plus particulièrement l'implantation des relations entre hypothèses laisse présager d'un avenir prometteur pour les liens en tant qu'outils de raisonnement.

- Enfin la mise en oeuvre du tutoriel et du système de gestion de la cohérence pour Smeci, en cours de développement, devrait renforcer la puissance des liens par l'ampleur de la tâche

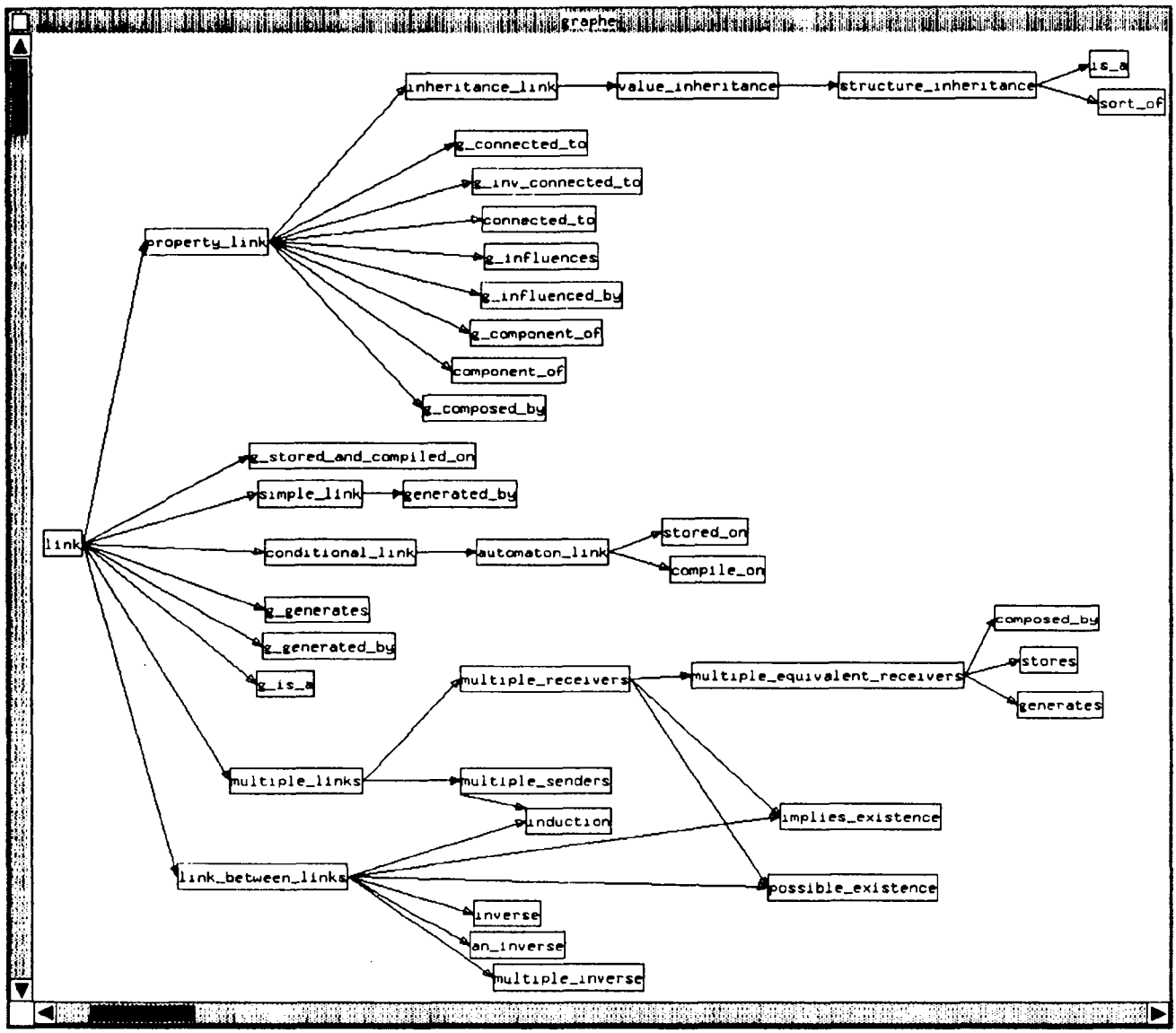


Figure 5.5: Ensemble des liens utilisés pour la réalisation du tutoriel.

conférée à ce mécanisme.

Chapitre 6

Conclusion

En facilitant l'expression des connaissances relationnelles entre les objets et leur maintien, nous étendons la puissance du modèle objet. Les relations entre les objets ne sont plus prises en compte par des mécanismes annexes, mais sont intégrées au système objet. Il en découle une plus grande flexibilité tant des liens que des objets; ils se définissent mutuellement.

L'implantation des relations dans Othelo valide le modèle de lien présenté. Le programmeur peut exploiter la hiérarchie de classes de lien proposée. Pour définir de nouvelles relations, il crée de nouvelles sous-classes puis les relations individuelles correspondant au problème à traiter. Cette représentation des relations présente plusieurs avantages :

- Dans notre modèle, la cohérence des relations individuelles est directement gérée par les relations. Son contrôle n'est donc ni centralisé dans un gestionnaire de cohérence, ni dispersé dans les objets. La sémantique associée aux relations est ainsi exprimée dans la relation.

- Toutes les informations inhérentes à une relation sont regroupées au niveau de la classe qui la représente, ce qui en facilite la compréhension et la modifiabilité.

- Une même relation peut être établie entre des objets de classes différentes. Une telle liberté peut être limitée en précisant au niveau de la relation les classes des objets qu'elle peut lier.

- Cette représentation des relations par classes respecte le principe d'encapsulation des données, puisque toute communication entre les objets est toujours réalisée par envoi de message. L'objet relation est une interface entre les objets liés.

- La détection des cycles peut être mise en oeuvre directement au niveau des relations par compteurs ou interruptions.

- La définition de nouvelles sortes de relations nécessite seulement la création dynamique de nouvelles classes.

- Si dans le langage objet considéré un attribut est un objet, il sera possible de lier directement deux attributs.

- La connaissance est modulaire. La vision du monde objet est structurée, d'un côté l'ensemble des objets du monde réel et de l'autre l'ensemble des objets relations.

La puissance du concept de lien a naturellement conduit à son utilisation dans plusieurs applications dont la gestion d'une interface cohérente pour Othelo, la création de liens d'héritage, la manupulation d'hypothèses et la réalisation d'un tutoriel. D'autre part une étude favorable a été faite sur leur utilisation, dans la conception d'un ATMS pour Smeci [Bou89].

Nous étudions actuellement les aspects formels du modèle de lien. En effet, celui-ci décrit le mécanisme des relations et conduit à une formalisation des relations de dépendance. Une application de cette étude serait de proposer une formalisation de l'héritage dans les langages à objets et de situer notre approche par rapport à Ducournau [DH87] pour la sémantique de l'héritage et Cardelli [Car84] pour la comparaison entre le typage et l'héritage. En faisant une analogie avec les travaux effectués autour des grammaires attribuées, des propriétés mathématiques pourraient être déduites et donneraient ainsi une dimension supplémentaire à cette approche. En effet, il serait intéressant d'étudier les avantages de compiler un graphe de dépendances pour déduire certaines propriétés sur les liens dont la détection statique de cycles.

Bibliographie

[Aid89] *Aida, Environnement de développement d'applications.* Ilog, 1989. Version 1.3.

[Ame87] P. America. Inheritance and subtyping in a parallel object oriented language. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 281–289, Paris, 1987.

[BC87] E. Blake and S. Cook. On including part hierarchies in object-oriented languages, with an implementation in Smalltalk. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 45–54, Juin 1987.

[Ber88] P. Berlandier. *Intégration d'outils pour l'expression et la satisfaction de contraintes dans un générateur de systèmes experts.* Rapports de Recherche 924, INRIA-SOPHIA ANTIPOLIS, Novembre 1988. Programme 1.

[Bor86] Alexender Borgida. Exceptions in object-oriented languages. *Sigplan Notices*, 21(10):107–119, October 1986.

[Bou89] J. Boursin. *Raisonnement Hypothétique en SMECI.* Rapport de Stage, DEA Intelligence Artificielle et Application de CAEN, Septembre 1989.

[BPR88] M.R. Blaha, W.J. Premerlani, and J.E. Rumbaugh. Relationnal database design using an object-oriented methodology. *Communications of the ACM*, 31(4):414–427, Avril 1988.

[Bra83] R.J. Brachman. What is-a is and isn't : an analysis of taxonomic links in semantic networks. In *Computer Knowledge Representation, IEEE*, pages 37–41, Octobre 1983.

[BS83] D.G. Bobrow and M. Stefik. *The LOOPS Manual.* 1983.

[Car84] L. Cardelli. A semantic of multiple inheritance. In *Lectures Notes in Computers Science, Semantics of data types*, Springer-Verlag, New-York, 1984.

[Cla85] B.D. Clayton. *Art, Programming tutorial.* Mars 1985.

[Coi87] P. Cointe. Metaclasses are first class : the ObjVlisp model. In *OOPSLA'87 Proceedings*, pages 156–165, October 1987.

[Col89] A. Colmerauer. Une introduction à Prolog III. In *Etat de l'art et perspectives en Programmation en logique, journée de synthèse*, pages 129–156, Afcet – INRIA, Paris, Janvier 1989.

[DG87] L.G. Demichiel and R.P. Gabriel. The common Lisp object system : an overview. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 201–222, Paris, June 1987.

[DH87] R. Ducournau and M. Habib. On some algorithms for multiple inheritance in object-oriented programming. In *ECOOP'87, European Conference on Object-Oriented Programming*, pages 291–302, Paris, 1987.

[DH89] R. Ducournau and M. Habib. La multiplicité de l'héritage dans les langages à objets. *TSI*, 8(1):41–62, Janvier 1989. Numéro Spécial Langages orientés objet.

[DP87] F. Dery and A.M. Pinna. Intégration à un système-expert d'outils graphiques pour la visualisation et le pilotage. In *Congrès RFIA: Reconnaissance des Formes et Intelligence Artificielle*, AFCET, Antibes, France, Novembre 1987.

[Dug87] P. Dugerdil. Les mécanismes d'héritage d'OBJLOG : vertical et sélectif multiple avec point-de-vue. In *Reconnaissance des formes et intelligence artificielle*, Antibes, France, November 1987.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept : an extension to the entity-relationship model. In *Data and Knowledge Engineering*, pages 75–116, 1985.

[FP90a] M. Fornarino and A.M. Pinna. *Un modèle objet logique et relationnel. Le langage Othelo.* PhD thesis, Université de Nice, Avril 1990.

[FP90b] MIreille Fornarino and Anne-Marie Pinna. *Maquette d'une MODELOTHEQUE. Acquisition de modles et raisonnement par assemblage.* Technical Report MGL/90-1165, CSTB, Octobre 1990.

[FTK*84] K. Furukawa, A. Takeuchi, S. Kunifuji, H. Yasukawa, M. Ohki, and K. Ueda. Mandala : a logic based knowledge programming system. In ICOT, editor, *Proceedings of the international conference on fifth Generation Computer System*, pages 613–622, Japan, 1984.

[Gro88] The CHIP Group. *CHIP Version 2.1.* European Computer-Industry Research Centre GmbH, West Germany, 1988.

[HM87] D. Herin-Aime and O. Massiot. Demsi : un prototype de système expert orienté objet pour l'évolution des systèmes d'information. In *Proceedings Cognitiva*, pages 237–241, La-Vilette Paris France, Mai 1987. Tome 1.

[IK87] H. Iline and H. Kanoui. Extending logic programming to object programming : the system lap. In *Proceedings of the tenth international joint Conference on Artificial Intelligence IJCAI'87*, pages 34–39, Milan, Italie, August 1987. tome 1.

[KEE85] *KEE v.2. Software Development System, User's Manual.* Intellicorp., 1985.

[Kos87] Y. Koseki. Amalgamating multiple programming paradigms in prolog. In *Proceedings of IJCAI*, pages 76–82, Milan, Italie, August 1987. Tome 1.

[LeL89] *Le-Lisp de l'INRIA Version 15.22 Le Manuel de référence.* Ilog, 1989.

[LTAZ87] JP. Laurent, F. Thome, J. Ayel, and D. Ziebelin. Kee, knowledge craft et art. evaluation comparative de trois outils de développement de systèmes experts. *Revue d'Intelligence Artificielle*, 1(2):25–53, 1987.

[MK87] M. Murata and K. Kusumoto. *Daemon: A Mediator that Keeps Wholes Consistent with their parts.* Technical Report, Fuji Xerox, 1987.

[RM90] M. Rueher and C. Michel. Using objects evolution for software processes representation. In IEEE Computer Society Press, editor, *Proc. Software Track*, pages 121–130, Hawai, 1990.

[SB86] M. Stefik and D.G. Bobrow. Object oriented programming : themes and variations. *AI Magazine*, 6(4):40–62, 1986.

[SME88] *SMECI Manuel de référence.* Ilog, 1.4 edition, 1988.

[Sny86] A. Snyder. Encapsulation and inheritance in object-oriented programming languages. In *OOPSLA'86 Proceedings*, pages 38–45, ACM, September 1986.

[Tro88] B. Trousse. Bénéfices d'une approche orientée objet pour un environnement de CAO. In *Proceedings of the MICAD 88*, pages 313–328, Paris, Mars 1988.

[Voy89] R. Voyer. *Oks: un langage de programmation par réflexes.* Technical Report 89/47, LAFORIA, 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399